IDIAP RESEARCH REPORT

RECURRENT CONVOLUTIONAL NEURAL NETWORKS FOR SCENE LABELING

Pedro H. O. Pinheiro Ronan Collobert

Idiap-RR-41-2013

DECEMBER 2013

Centre du Parc, Rue Marconi 19, P.O. Box 592, CH - 1920 Martigny
T +41 27 721 77 11 F +41 27 721 77 12 info@idiap.ch www.idiap.ch

Recurrent Convolutional Neural Networks for Scene Labeling

Pedro O. Pinheiro^{1,2}**, Ronan Collobert**^{1}
^{1}Idiap Research Institute, Martigny, Switzerland
^{2}Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
{pedro.pinheiro, ronan.collobert}@idiap.ch

Abstract

Scene labeling is a technique that consist on giving a label to every pixel in an image according to the class they belong to. To ensure a good visual coherence and a high class accuracy, it is essential for a scene parser to capture long range dependencies on images. In a *feed-forward* architecture, this can be simply achieved by considering a sufficiently large input context patch, around each pixel to be labeled. We propose an approach consisting of a recurrent convolutional neural network which allows us to consider a large input context, while limiting the capacity of the model. Contrary to most standard approaches, our method does not rely on any segmentation methods, nor any task-specific features. The system is trained in an end-to-end manner over raw pixels, and models complex spatial dependencies with low inference cost. As the context size increases with the built-in recurrence, the system identifies and corrects its own errors. Our approach yields state-of-the-art performance on both the Stanford Background Dataset and the SIFT Flow Dataset, while remaining very fast at test time.

1 Introduction

In the computer vision field, *scene labeling* is the task of fully labeling an image pixel-by-pixel with the class of the object each pixel belongs to. This task is very challenging, as it implies solving a detection, a segmentation and a multi-label recognition problem all in one.

The scene labeling problem is most commonly addressed with some kind of *local* classifier constrained in its predictions with a graphical model (*e.g.* Conditional Random Fields, Markov Random Fields), in which *global* decisions are made. These approaches usually consist of segmenting the image into superpixels or segment regions to assure a visible consistency of the labeling and also to take into account similarities between neighbor segments, giving a high level understanding of the overall structure of the image. Each segment contains a series of input features describing it and contextual features describing spatial relation between the label of neighbor segments. These models are then trained to maximize the likelihood of correct classification given the features [1, 2, 3, 4, 5, 6, 7]. The main limitation of scene labeling approaches based on graphical models is the computational cost at test time, limiting the model to simple contextual features.

In this work, we consider a *neural network-based feed-forward* approach which can take into account long range dependencies on the image while controlling the capacity of the network, achieving state-of-the-art accuracy while keeping the computational cost low at test time, thanks to the complete feed-forward design. Our method relies on a recurrent architecture for convolutional neural networks: a sequential series of networks sharing the same set of parameters. Each instance consider as input both an RGB image and the classification attempt of the previous instance of the network. The network learns itself how to smooth its own predicted labels, improving the estimation as the number of instances increases.

Table 1: Comparison between different methods for scene labeling. The advantage of our proposed method consists on the simplicity of inference, not relying on any task-specific feature extraction nor segmentation method.

METHOD	TASK-SPECIFIC FEATURES
GOULD ET AL., 2009 [1]	17-DIMENSIONAL COLOR AND TEXTURE FEATURES, 9 GRID LOCATIONS AROUND THE PIXEL AND THE IMAGE ROW, REGION SEGMENTATION.
MUNOZ ET AL., 2010 [8]	GIST, PYRAMID HISTOGRAM OF ORIENTED GRADIENTS, COLOR HISTOGRAM CIELAB, RELATIVE RELOCATION, HIERARCHICAL REGION REPRESENTATION.
KUMAR & KOLLER, 2010 [2]	COLOR, TEXTURE, SHAPE, PERCENTAGE PIXELS ABOVE HORIZONTAL, REGION-BASED SEGMENTATION.
SOCHER ET AL., 2012 [6]	SAME AS [1].
LEMPITSKY ET AL., 2011 [7]	HISTOGRAM OF VISUAL SIFT, HISTOGRAM OF RGB, HISTOGRAM OF LOCATIONS, "CONTOUR SHAPE" DESCRIPTOR.
TIGHE & LAZEBNIK, 2010 [3]	GLOBAL, SHAPE, LOCATION, TEXTURE/SIFT, COLOR, APPEARANCE, MRF.
FARABET ET AL., 2013 [9]	LAPLACIAN PYRAMID, SUPERPIXELS/CRF/TREE SEGMENTATION.
OUR RECURRENT CNN	RAW PIXELS

Compared to graphical models approaches relying on image segmentation, our system has several advantages: (i) it does not require any engineered features, since deep learning architectures train (hopefully) adequate discriminative filters in an end-to-end manner, (ii) the prediction phase does not rely in any label space searching, since it requires only the *forward evaluation of a function*.

2 Related Work

In a preliminary work, [10] proposed an innovative approach to scene labeling without the use of any graphical model. The authors propose a solution based on deep convolutional networks relying on a *supervised* greedy learning strategy. These network architectures can be fed with raw pixels and are able to capture texture, shape and contextual information.

[6] also considered the use of deep learning techniques to deal with scene labeling. Unlike us, the authors consider off-the-shelf features of segments obtained from the scenes. They then use a network for recursively merging different segments and give them a semantic category label. Our recurrent architecture differs from theirs in the sense that we use it to parse the scene with a smoother class annotation.

More recently, [9] also consider the use of convolutional networks, extracting features densely from a multiscale pyramid of images. This solution yields satisfactory results for the categorization of the pixels, but poor visual coherence.

The authors propose three different over-segmentation approaches to produce the final labeling with improved accuracy and better visual coherence: (i) the scene is segmented in superpixels and a single class is assigned to each of the superpixels, (ii) a conditional random field is defined over a set of superpixels to model joint probabilities between them and correct aberrant pixel classification (such as "road" pixel surrounded by "sky") and (iii) the selection of a subset of tree nodes that maximize the average "purity" of the class distribution, hence maximizing the overall likelihood that each segment will contain a single object. The superpixel-based approach was however order of magnitude faster than the two other approaches, with slightly lower performance in accuracy. In contrast, our approach is simpler and completely feed-forward, as it does not require any image segmentation technique, nor the handling of a multiscale pyramid of input images.

As in [9], [11] proposed a similar multiscale convolutional architecture. In their approach, the authors smooth out the predicted labels with pairwise class filters.

Compared to existing approaches, our method does not rely on any task-specific feature (see Table 1). Our scene labeling system is able to extract relevant contextual information from raw pixels.

3 Systems Description

We formally introduce convolutional neural networks (CNNs) in Section 3.1. In Section 3.2 we discuss how to capture long range dependencies with these type of models, while keeping a tight control on the capacity. Section 3.3 introduces our recurrent network approach for scene labeling.

3.1 Convolutional Neural Networks for Scene Labeling

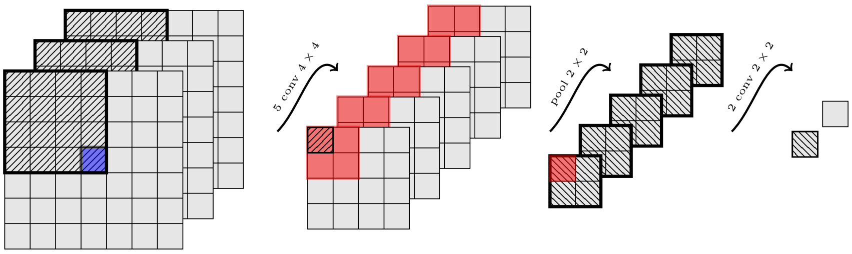


Figure 1: A simple convolutional network. Given an image patch providing a context around a pixel to classify (here blue), a series of convolutions and pooling operations (filters slid through input planes) are applied (here 5 4×4 convolutions, followed by a 2×2 pooling, followed by 2 2×2 convolutions. Each 1×1 output plane is interpreted as a score for a given class.

Convolutional neural networks [12] are a natural extension of neural networks for treating images. Their architecture, vaguely inspired by the biological visual system, possesses two key properties that make them extremely useful for image applications: spatially shared weights and spatial pooling. These kind of networks learn features that are shift-invariant, *i.e.*, filters that are useful across the entire image (due to the fact that image statistics are stationary). The pooling layers are responsible for reducing the sensitivity of the output to slight input shift and distortions. This type of neural network has proven to be very efficient in many vision applications, such as object recognition and segmentation ([13, 14]).

A typical convolutional network is composed of multiple stages, as shown on Figure 1. The output of each stage is made of a set of two dimensional arrays called feature maps. Each feature map is the outcome of one convolutional (or pooling) filter applied over the full image. A non-linear squashing function (such as a hyperbolic tangent) always follows a pooling layer.

In the context of scene labeling we consider a set of images indexed by I_k, and we are interested in finding the label of each pixel at location (i, j), for every image k. To that matter, the network is fed with a squared *context patch* $I_{i,j,k}$ surrounding the pixel at location (i, j) in the k-th image. It can be shown (see Figure 1) that the output plane size sz_m of the m^{th} layer is computed as:

$$sz_m = \frac{sz_{m-1} - kW_m}{dW_m} + 1 \,, \tag{1}$$

where sz_0 is the input patch size, kW_m is the size of the convolution (or pooling) kernels in the m^{th} layer, and dW_m is the pixel step size used to slide the convolution (or pooling) kernels over the input planes.[1] Given a network architecture and an input image, one can compute the output image size by successively applying Equation 1 on each layer of the network. During the training phase, the size of the input patch $I_{i,j,k}$ is chosen carefully such that the output layers produces 1×1 planes, which are then interpreted as scores for each class of interest.

Adopting the same notation as [9], the output of a network f with M stages and trainable parameters $(\mathbf{W}, \mathbf{b})$, given an input patch $I_{i,j,k}$, can be formally written as:

$$f(I_{i,j,k}; (\mathbf{W}, \mathbf{b})) = \mathbf{W}_M \mathbf{H}_{M-1} \,, \tag{2}$$

[1] Most people use $dW = 1$ for convolutional layers, and $dW = kW$ for pooling layers.

with the output of the m^{th} hidden layer computed as:

$$\mathbf{H}_m = \tanh(\text{pool}(\mathbf{W}_m \mathbf{H}_{m-1} + \mathbf{b}_m)) \,, \tag{3}$$

for $m = \{1, ..., M\}$ and denoting $\mathbf{H}_0 = I_{i,j,k}$. $\mathbf{b}_m$ is the bias vector of layer m and $\mathbf{W}_m$ is the Toeplitz matrix of connection between layer $m - 1$ and layer m. The pool$(\cdot)$ function is the max-pooling operator and $\tanh(\cdot)$ is the point-wise hyperbolic tangent function applied at each point of the feature map.

In order to train the network by maximizing a likelihood, the network scores $f_c(I_{i,j,k}; (\mathbf{W}, \mathbf{b}))$ (for each class of interest $c \in \{1, ..., N\}$) are transformed into conditional probabilities, by applying a *softmax* function:

$$p(c|I_{i,j,k}; (\mathbf{W}, \mathbf{b})) = \frac{e^{f_c(I_{i,j,k}; (\mathbf{W}, \mathbf{b}))}}{\sum\limits_{d \in \{1,...,N\}} e^{f_d(I_{i,j,k}; (\mathbf{W}, \mathbf{b}))}} \tag{4}$$

The parameters $(\mathbf{W}, \mathbf{b})$ are learned in an end-to-end supervised way, by minimizing the negative log-likelihood over the training set:

$$L(\mathbf{W}, \mathbf{b}) = -\sum_{I_{(i,j,k)}} \ln p(l_{i,j,k}|I_{i,j,k}; (\mathbf{W}, \mathbf{b})) \,, \tag{5}$$

where $l_{i,j,k}$ is the correct pixel label class, at position (i, j) in image I_k. The minimization was achieved with the Stochastic Gradient Descent (SGD) algorithm, with a fixed learning rate λ:

$$\mathbf{W} \longleftarrow \mathbf{W} - \lambda \frac{\partial L}{\partial \mathbf{W}} \;;\; \mathbf{b} \longleftarrow \mathbf{b} - \lambda \frac{\partial L}{\partial \mathbf{b}} \,. \tag{6}$$

Finally, in test time, for every test image I, the pixel at location (i, j) of image k is labeled with the *argmax* of the network predictions:

$$\hat{l}_{i,j,k} = \underset{c \in \text{classes}}{\arg\max}\, p(c|I_{i,j,k}; (\mathbf{W}, \mathbf{b})) \,, \tag{7}$$

considering the context patch $I_{i,j,k}$. Note that this might imply adding padding when inferring label of pixels close to the image border.

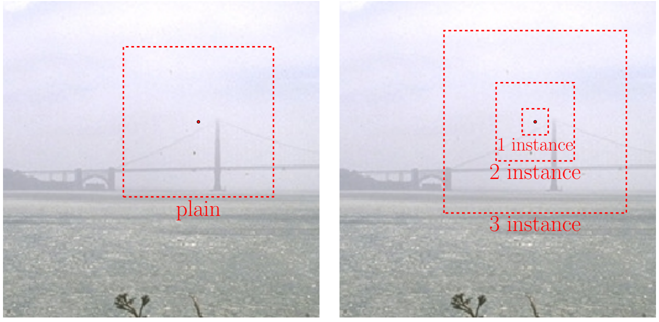
3.2 Long Range Dependencies with Convolutional Networks

Existing successful scene labeling systems leverage long range image dependencies in some way. The most common approach is to add some kind of graphical model (*e.g.* a conditional random field) over local decisions, such that a certain global coherence is maintained. In the case of convolutional networks, an obvious way to capture long range dependencies efficiently would be to consider large input patches when labeling a pixel. Unfortunately, this approach might face generalization issues, as considering larger context often implies considering larger models (*i.e.* higher capacity).

In Table 2, we review possible ways to control the capacity of a convolutional neural network, assuming a large input context. In a “plain” architecture (as described in Section 3.1), one can easily control the capacity by increasing the filter sizes in pooling layers, reducing the overall number of parameters in the network. Unfortunately, performing large poolings decreases the network label output resolution (*e.g.*, if one performs a $1/8$ pooling, the label output plane size will be about $1/8^{th}$ of the input image size). One can overcome this problem, but at the cost of a slow inference process.

Instead, [9] considered the use of a *multiscale* convolutional network. Large contexts are integrated into local decisions while making the model still manageable in terms of parameters/dimensionality. Label coherence is then increased by leveraging superpixels.

Another way to consider a large input context size while controlling the capacity of the model is to make the network recurrent: the output of the model is fed back to the input of another instance of



(a) (b)

Figure 2: Context input patch of "plain" (a) and recurrent (b) architecture. The size of the contextual input patch (b) increases as the number of instances in the recurrent convolutional network increases. The capacity of the model remains the same, since the parameters over all instances are shared.

Table 2: Long range dependencies integration in CNN-based scene labeling systems. Ways to control capacity and speed of each architecture is reported.

MEANS	CAPACITY CONTROL	SPEED
LOCAL CLASSIFIER + GRAPHICAL MODEL	–	SLOW
MULTISCALE	SCALE DOWN INPUT IMAGE	FAST
LARGE INPUT PATCHES	INCREASE POOLING	SLOW
	RECURRENT ARCHITECTURE	FAST

the same network, which *shares* the same parameters (see Figure 3). Given Equation 1, we have

$$sz_{m-1} = dW_m \times sz_m + (kW_m - dW_m)\,.$$

Thus, the required context to label one pixel (*i.e.* if the network output size is 1×1), increases with the number of network instances (see an example in Figure 2). However, the capacity of the system remains constant, since the parameters of each network instance are simply shared. We will now detail our recurrent network approach.

3.3 Recurrent Network Approach

The recurrent architecture (see Figure 3) consists of P instances of the "plain" convolutional network $f(\cdot)$, each of them with *identical* (shared) trainable parameters $(\mathbf{W}, \mathbf{b})$. Each instance f^p $(1 \leq p \leq P)$ is fed with an input "image" $\mathbf{F}^p$ of $N+3$ features maps

$$\mathbf{F}^p = [f^{p-1}(I^{p-1}_{i,j,k}; (\mathbf{W}, \mathbf{b})), I^p_{i,j,k}], \qquad \mathbf{F}^1 = [\mathbf{0}, I_{i,j,k}].$$

which are the output label planes of the previous instance, and the scaled[2] version of the raw RGB squared patch surrounding the pixel at location (i, j) of the training image k. To make the number of inputs equal in all instances of the network, the input "image" of the first instance is simply the patch of raw pixel coupled with N 0 feature maps. This guarantees the end-to-end characteristics of the system as well as its fast inference during test.

As in the "plain" network, the size of the patch during training is chosen such that the output layers produces 1×1 planes, which are then interpreted as scores for each class of interest.

Besides the obvious advantage of capturing long range dependencies while controlling the capacity, the recurrent architecture also the quality of achieving visually coherent results. Due to its recursivity, the system is able to learn to correct its mistakes done in previous instances.

[2] $I^{p-1}_{i,j,k}$ is scaled so that it has the width/height as f^{p-1}.

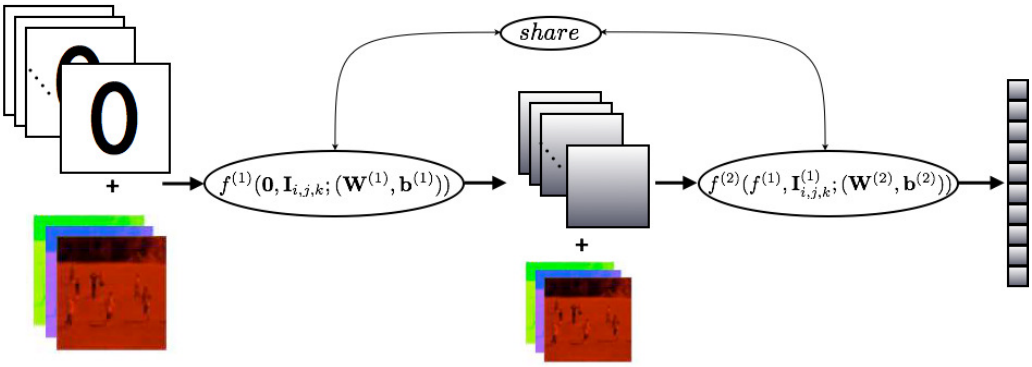

share
0
+
$f^{(1)}(\mathbf{0}, \mathbf{I}_{i,j,k}; (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}))$
+
$f^{(2)}(f^{(1)}, \mathbf{I}^{(1)}_{i,j,k}; (\mathbf{W}^{(2)}, \mathbf{b}^{(2)}))$

Figure 3: Recurrent network architecture with two instance of f. The first instance $f^{(1)}$ of the recurrent architecture is fed with a RGB patch and an empty feature map. The output of the first network is coupled with the scaled RGB patch and fed to the same network (shared parameters ($\mathbf{W}$ and $\mathbf{b}$)).

Table 3: Pixel and averaged per class accuracy and the computing time of other methods and our proposed approaches on the Stanford Background Dataset.

METHOD	PER PIXEL ACCURACY	AVG PER CLASS ACCURACY	COMPUTE TIME (S)
GOULD ET AL., 2009 [1]	76.4%	-	10 TO 600
TIGHE & LAZEBNIK, 2010 [3]	77.5%	-	10 TO 300
MUNOZ ET AL., 2010 [8][3]	76.9%	66.2%	12
KUMAR & KOLLER, 2010 [2]	79.4%	-	< 600
SOCHER ET AL., 2012 [6]	78.1%	-	?
LEMPITSKY ET AL., 2011 [7]	81.9%	72.4%	> 60
FARABET ET AL., 2013 [9][4]	78.8%	72.4%	0.6
FARABET ET AL., 2013 [9][5]	81.4%	76.0%	60.5
PLAIN CNN	79.4%	69.5%	15
CNN_1	67.9%	58.0%	0.2
rCNN_1 (2 INSTANCES)	79.5%	69.5%	2.6
CNN_2	15.3%	14.7%	0.06
rCNN_2 (2 INSTANCES)	76.2%	67.2%	1.1
rCNN_2 (3 INSTANCES)	79.8%	69.3%	2.15

4 Experiments

We tested our proposed method on two different datasets for scene labeling: the Stanford Background [1] and the SIFT Flow Dataset [5]. The Stanford dataset has 715 images from rural and urban scenes composed of 8 classes. The scenes have approximately 320×240 pixels. As in [1], we performed a 5-fold cross-validation with the dataset randomly split into 572 training images and 143 test images in each fold. The SIFT Flow is a larger dataset composed of 2688 images of 256×256 pixels and 33 semantic labels. All the algorithms and experiments were implemented using Torch7 [15].

Each image on the training set was properly padded and normalized such that they have zero mean and unit variance. All networks were trained by sampling patches surrounding randomly chosen pixel from randomly chosen images from the training set of images. Contrary to [9] (i) we did not

[3] Unpublished improved results have been recently reported by the authors.

[4] Multiscale CNN + superpixels.

[5] Multiscale CNN + CRF.

Table 4: Pixel and averaged per class accuracy of other methods and our proposed approaches on the SIFT Flow Dataset.

METHOD	PER PIXEL ACCURACY	AVG PER CLASS ACCURACY
LIU ET AL., 2009 [5]	74.75%	-
TIGHE & LAZEBNIK, 2010 [3]	77.0%	30.1%
FARABET ET AL., 2013 [9]	78.5%	29.6%
PLAIN	76.5%	30.0%
CNN_1	51.8%	17.4%
rCNN_1	76.2%	29.2%
rCNN_2 (2 INSTANCES)	65.5%	20.8%
rCNN_2 (3 INSTANCES)	77.7%	29.8%

consider any extra distortion on the images[6], and (ii) we did not sample training patches according to balanced class frequencies.

We considered two different accuracy measures to compare the performance of our proposed methods with other approaches. The first one is the accuracy per pixel of test images. This measure is simply the ratio of correct classified pixels of all images in the test set. However, in scene labeling (especially in datasets with large number of classes), classes which are much more frequent than others (*e.g.* the "sky" class is much more frequent than "moon") have more impact on this measure. Recent papers also consider the averaged per class accuracy on the test set (all classes have the same weight in the measure). Note that as mentioned above, we did not train with balanced class frequencies, which would have optimized this second measure.

We consider a "plain" architecture ("plain CNN") with a large patch and strong number of pooling and two recurrent architectures with two ("rCNN_1") and three ("rCNN_2") instances. In this section, rCNN_i represents a recurrent version of the regular convolutional network CNN_i.

Table 3 compares the performance of our architecture with related works on the Stanford Background Dataset and Table 4 compares the performance on the SIFT Flow Dataset. Note that the inference time in the second dataset does not change, since we exclude the need of any segmentation method.

In the following, we provide additional technical details for each architecture used.

4.1 Plain Network

The first "plain" network was trained with 133×133 input patches. The network was composed of a 6×6 convolution with nhu_1 output planes, followed by a 8×8 pooling layer, a $\tanh(\cdot)$ non-linearity, another 3×3 convolutional layer with nhu_2 output planes, a 2×2 pooling layer, a $\tanh(\cdot)$ non-linearity, and a final 7×7 convolution to produce label scores. The hidden units were chosen to be $nhu_1 = 25$ and $nhu_2 = 50$ for the Stanford dataset, and $nhu_1 = 50$ and $nhu_2 = 50$ for the SIFT Flow dataset.

4.2 Recurrent Architectures

We consider two different recurrent convolutional network architectures.

The first architecture, rCNN_1, is composed of two consecutive instances of the convolutional network CNN_1 with shared parameters (as in Figure 3). CNN_1 is composed of a 8×8 convolution with 25 output planes, followed by a 2×2 pooling layer, a $\tanh(\cdot)$ non-linearity, another 8×8 convolutional layer with 50 output planes, a 2×2 pooling layer, a $\tanh(\cdot)$ non-linearity, and a final 1×1 convolution to produce N label scores.

rCNN_1 is trained by considering the two network instances simultaneously. For each training example we randomly choose to perform a "forward" and "backward" pass through one or two instances of the network. This training approach allows the network to learn to correct its own mistakes (made by the first network instance). As mentioned in Section 3.2, the input context patch size depends

[6] Which is known to improve the generalization accuracy by few extra percents.

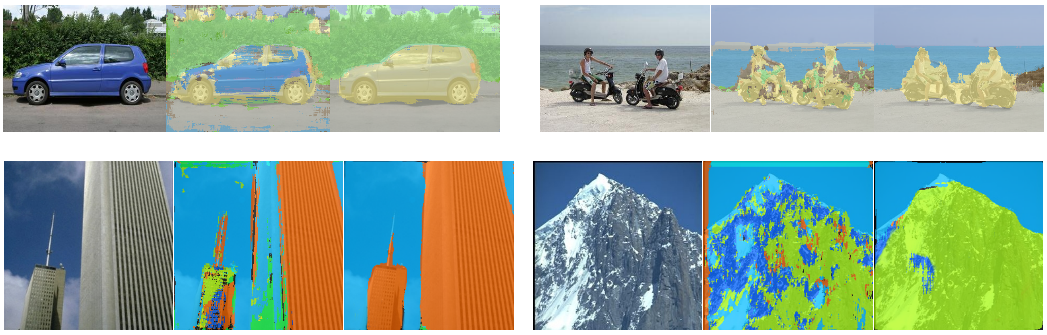


Figure 4: Inference results of the recurrent system. The first line shows two examples from the Stanford Background Dataset and the second examples from the SIFT Flow dataset. For each example, the first column represents the original RGB image, middle one illustrates results of rCNN$_2$ assuming two instances and the last one the result assuming three instances: most mistakes of second instance are corrected on the third one.

directly on the number of network instances in the recurrent architecture. In the case of rCNN$_1$, the patch size is of 25×25 when considering one instance and 121×121 when considering two network instances.

The second recurrent convolutional neural network, rCNN$_2$, is composed of three instances of the convolutional network CNN$_2$, with shared parameters. Each instance of CNN$_2$ is composed of a 8×8 convolution with 25 output planes, followed by a 2×2 pooling layer, a $\tanh(\cdot)$ non-linearity, another 8×8 convolution with 50 planes and a final 1×1 convolution which outputs the N label planes.

In rCNN$_2$, the first two instances are trained simultaneously[7] (as in Figure 3) through SGD, with input patch of size 67. Then, after the system with two instances are trained, a third instance of the network is considered (still with the parameters shared with the others instances) so that it is able to correct itself from the previous labeling. The input patch is of size 155 in this latter case.

Figure 4 illustrates inference of the recurrent network with two and three instances. The network learns itself how to correct its own label prediction.

In all cases, the learning rate in Equation 6 was equal to 10^{-4}. All hyper-parameters were tuned with a 10% hold-out for validation.

5 Conclusion

This paper presents a *feed-forward* approach for scene labeling, based on supervised "deep" learning strategies which models in a rather simple way non-local class dependencies in a scene from raw pixels. We demonstrate that the problem of scene labeling can be faced without the need of any expensive graphical model or segmentation tree technique to ensure labeling. The scene labeling is inferred simply by forward evaluation of a function applied to a RGB image.

In terms of accuracy, our system achieved state-of-the-art results on both Stanford Background and SIFT Flow dataset, while keeping a fast inference time. Future work includes investigation of unsupervised or semi-supervised pre-training of the models, as well as application to larger datasets such as the Barcelona dataset.

Acknowledgments

This work was supported by the Swiss NSF through the Swiss National Center of Competence in Research (NCCR) on Interactive Multimodal Information Management (www.im2.ch).

[7] Considering only one instance in this case produces a very small context input patch.

References

[1] S. Gould, R. Fulton, and D. Koller, "Decomposing a scene into geometric and semantically consistent regions," in *IEEE 12th International Conference on Computer Vision (ICCV)*, 2009, pp. 1–8.

[2] M. P. Kumar and D. Koller, "Efficiently selecting regions for scene understanding," in *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2010, p. 32173224.

[3] J. Tighe and S. Lazebnik, "Superparsing - scalable nonparametric image parsing with superpixels," *International Journal of Computer Vision*, vol. 101, no. 2, pp. 329–349, 2013.

[4] J. Verbeek and B. Triggs, "Scene segmentation with crfs learned from partially labeled images," in *Neural Information Processing Systems (NIPS)*, 2008.

[5] C. Liu, J. Yeun, and A. Torralba, "Nonparametric scene parsing: Label transfer via dense scene alignment," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 1972–1979.

[6] R. Socher, C. Lin, A. Ng, and C. Manning, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 2011.

[7] V. Lempitsky, A. Vedaldi, and A. Zisserman, "A pylon model for semantic segmentation," in *Neural Information Processing Systems (NIPS)*, 2011.

[8] D. Munoz, J. Bagnell, and M. Hebert, "Stacked hierarchical labeling," in *European Conference on Computer Vision (ECCV)*, 2010.

[9] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013, in press.

[10] D. Grangier, L. Bottou, and R. Collobert, "Deep convolutional networks for scene parsing," in *International Conference on Machine Learning (ICML) Deep Learning Workshop*, 2009.

[11] H. Schulz and S. Behnke, "Learning object-class segmentation with convolutional neural networks," in *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, 2012.

[12] Y. LeCun, "Generalization and network design strategies," in *Connectionism in Perspective*, R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, Eds. Elsevier, 1989.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems (NIPS 1989)*, D. Touretzky, Ed., vol. 2. Denver, CO: Morgan Kaufman, 1990.

[14] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proc. International Conference on Computer Vision (ICCV'09)*. IEEE, 2009.

[15] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Implementing neural networks efficiently," in *Neural Networks: Tricks of the Trade*, G. Montavon, G. Orr, and K.-R. Muller, Eds. Springer, 2012.